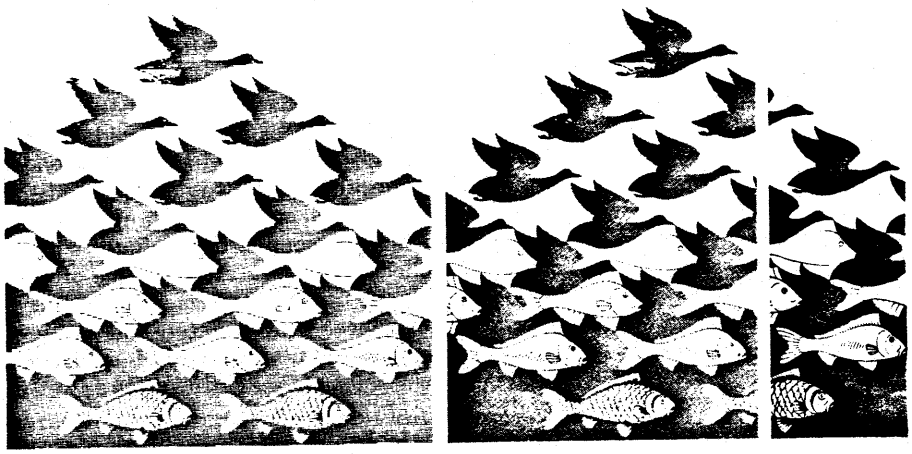
Benchmarking is older. Simulation is more popular. But analytical queueing models may offer the most cost-effective technique for computer system performance modeling.



Queueing Models of Computer Systems

Arnold O. Allen
IBM Systems Science Institute

Figure 1 illustrates the spectrum of techniques that have been used for computer system performance modeling. The leftmost technique, the use of simple rules of thumb, is the easiest to apply and also requires the smallest investment of resources, while the rightmost technique, benchmarking, is the most difficult and costly.

The rules of thumb range from simple, well-known observations such as Murphy's law and Parkinson's law to some laws cited by Dickson[1] which have obvious applications to computer system performance evaulation:

Evie Nef's law. There is a solution to every problem; the only difficulty is finding it.

Unnamed law. If it happens, it must be possible.

Miller's law. You can't tell how deep a puddle is until you step into it.

Uhlmann's razor. When stupidity is a sufficient explanation, there is no need to have recourse to any other.

On a more prosaic level we have a few rules of thumb developed by experience and recorded by R. M. Schardt[2]:

(1) Generally, channel utilization in direct access storage devices should not exceed 35 percent for on-line applications or 40 percent for batch applications.
(2) Individual DASD device utilizations should not exceed 35 percent.
(3) Average arm seek time on a DASD device should not exceed 50 cylinders.
(4) No block size for either tape or disk should be less than 4K bytes.

These rules of thumb were selected for computers operating under control of the IBM MVS (multiple virtual storages) operating system, but many performance analysts would agree that they are good general rules.

Rules of thumb are excellent for day-to-day operation of computer installations but are not much help in predicting hardware and software upgrading requirements. Linear projection of utilization and load offers a more organized approach. It requires measuring and tracking of such items as central processor utilization, channel utilization, communication line utilization, etc., to establish growth rates for extrapolation. For reasonable effectiveness, the resource requirements of applications under development must also be estimated. This technique is certainly an improvement upon the rules-of-thumb technique and, if properly implemented, can be a helpful planning tool. A major problem, however, is that this approach attempts to use linear prediction for systems that are inherently nonlinear.

Simulation has been a very popular form of computer system modeling for years. It enables the analyst to model the system at a much greater level of

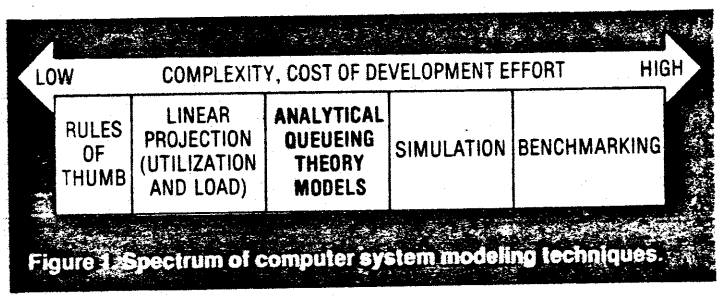


Figure 1. Spectrum of computer system modeling techniques.

0018-9162/80/0400-0013$00.75 © 1980 IEEE

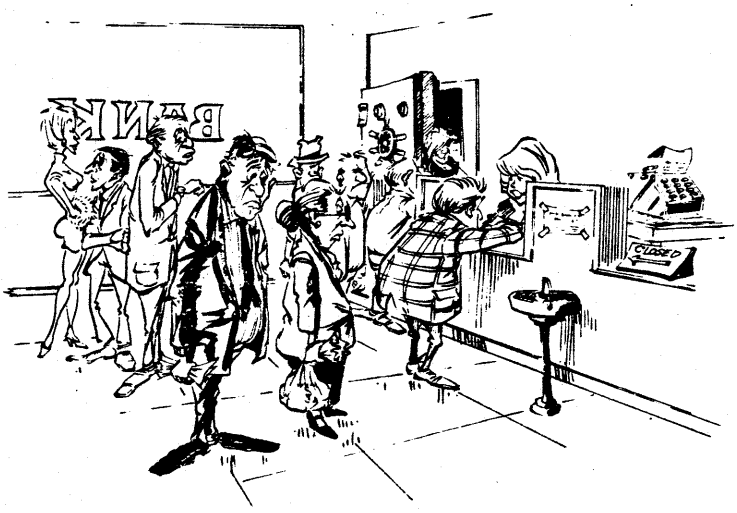


Figure 2. A typical queueing system. (Reprinted by permission of Xyzyx Information Corporation, Canoga Park, California.)

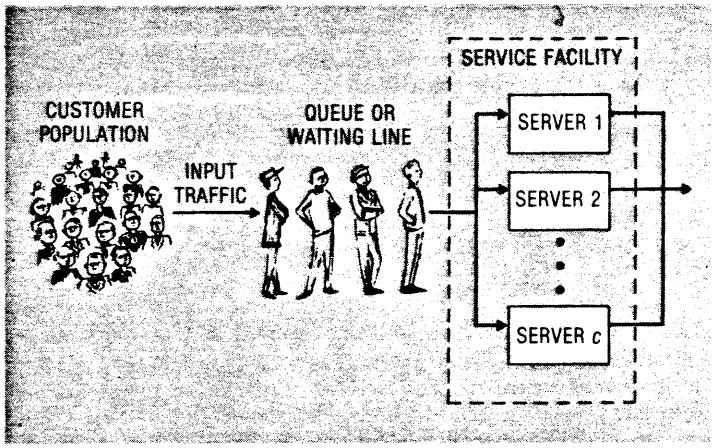


Figure 3. Elements of a queueing system. (© Academic Press, *Probability, Statistics, and Queueing Theory with Computer Science Applications,* 1978)

detail than is practicable with queueing theory models. Unfortunately, simulation models are difficult and costly to construct, validate, and run. Simulation projects tend to become large, long-term, and very difficult to control.

In terms of cost and complexity there is no sharp distinction between simulation and benchmarking. Some simulation models are as complex and hard to understand as the system being simulated! Likewise, a benchmark effort may be of rather limited scope, but if it is, it is probably nearly useless.

The benchmarking approach to performance evaluation is probably the oldest and most widely used technique, although its use has been primarily for new hardware selection. Thus, it is not a promising technique for deciding when a hardware update is required. The technique sounds disarmingly simple. You simply collect a representative sample of your workload, run it on the proposed machine, and measure the performance. Unfortunately, all three of these tasks are very difficult in practice. Lucas[3] gives an excellent discussion of these difficulties as well as a good explanation of simulation.

One of the major benchmarking problems of recent vintage has been that of generating the on-line component of the workload. In recent years this problem has been greatly simplified by the emergence of the type of program called a driver. This tool is exemplified by the IBM Teleprocessing Network Simulator.[4]

A driver is a simulator which uses the actual or planned user-specified data communication network (terminals, lines, etc.) to model the network and to generate and send messages to the computer system under test, with a specific message mix and rate for each terminal. Thus a driver can be used to approximate system performance and response times, evaluate communication network design, and test new application programs. It may be less complex to

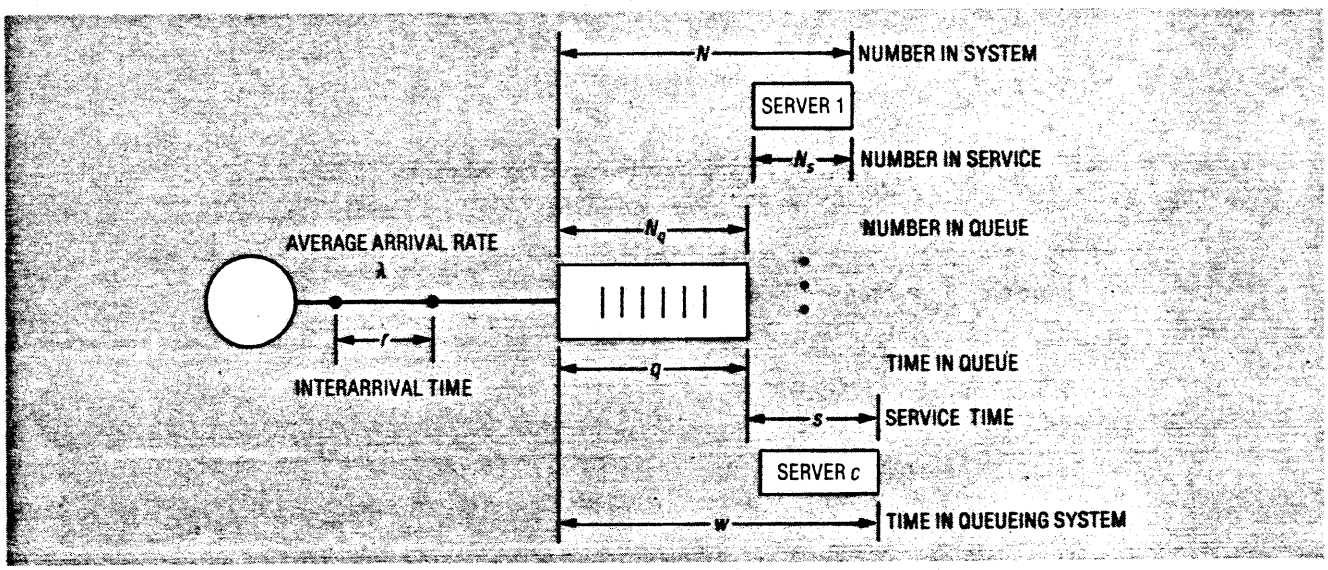


Figure 4. Some random variables used in queueing theory models. (© Academic Press, *Probability, Statistics, and Queueing Theory with Computer Science Applications,* 1978)

use than some detailed simulation models but is expensive in terms of hardware required.

In the last two or three years the use of analytical queueing models has become popular, although only a few years ago simulation was the prevailing technique. In 1978 an entire issue of *ACM Computing Surveys*[5] was devoted to queueing. Spragins[6] discusses the problems of complex systems modeling and cites a number of cases showing how analytical queueing system models that are much simpler than the system modeled can be used successfully. This article considers some analytical queueing theory models and shows how they can be used for performance modeling of computer systems. Most of the examples are drawn from another work,[7] where they are discussed in greater depth.

Elements of queueing theory

A queue is a waiting line, and queueing theory is the study of waiting-line phenomena. Figure 2 shows a typical queueing system; the poor befuddled fellow in the foreground has just discovered that the lady in front of him is the proprietor of the Junque Shop and is bringing in her weekly receipts. (He is waiting to rob the bank.)

In Figure 3 we show the elements of an open queueing system. There is a population or source of potential customers, where the term "customer" means an entity desiring some type of *service*—the transmission of a message, the processing of an inquiry, or the servicing of an I/O request—from a service facility. In the service facility, there are one or more *servers*, which are units that provide the required service for the customers. If all the servers are busy when a customer enters the system, the customer joins a queue until a server is available—if there is room in the waiting room.

Some random variables used in studying queueing systems are illustrated in Figure 4. (The reader will probably recall that a random variable is a variable that is not deterministic and thus must be described in probabilistic terms; that is, it has an associated probability distribution.) We use q to represent the time an arbitrary customer spends in the queue waiting for a server to become available (the queueing time) and s for the time required for the server to provide service (the service time); thus, w, the total time a customer spends in the queueing system, is given by $w = q + s$. Table 1 summarizes the queueing theory definitions used in this article. (With a few exceptions, the notations recommended in the *Queueing Standardization Conference Report* of May 11, 1971, issued by representatives of ORSA, AIIE, CORS, and TIMS, are followed.)

Specification of a queueing system

A mathematical study of a queueing system requires that we discuss the following queueing specifications.

Source. The population source can be finite or infinite. A finite source system cannot have an arbitrarily long queue for service, but the number of customers in the system affects the arrival rate. For an infinite source system the queue for service is unlimited, but the arrival rate is unaffected by the number of customers present in the system. If the source is finite but large, we assume an infinite customer population to simplify the mathematics. Buzen and Goldberg[8] offer some guidelines for choosing between infinite and finite models; computation is simpler for infinite models.

Arrival process. We assume that customers enter the queueing system at times $t_0 < t_1 < t_1 < t_2 \ldots t_n$. The random variables $\tau_k = t_k - t_{k-1}$ (where $k \geq 1$) are called *interarrival times*. We assume that the τ_k form a sequence of independent and identically distributed

Table 1.
Queueing theory notation and definitions.

$C(c,u)$	Erlang's C formula or the probability all c servers are busy in an $M/M/c$ queueing system.
$E[s]$	Expected (average or mean) service time for one customer.
$E[\tau]$	Expected (average or mean) interarrival time. $E[\tau] = 1/\lambda$ where λ is average arrival rate.
L	$E[N]$, expected (average or mean) number in the queueing system when the system is in the steady state.
L_q	$E[N_q]$, expected (average or mean) number in the queue, not including those in service, for steady state system.
λ	Average (mean) arrival rate to queueing system $\lambda = 1/E[\tau]$ where $E[\tau]$ = average interarrival time.
λ_T	Average throughput of a computer system measured in jobs or interactions per unit time.
μ	Average (mean) service rate per server. Average service rate $\mu = 1/E[s]$, where $E[s]$ is the average (mean) service time.
N	Random variable describing number in queueing system when system is in the steady state.
N_q	Random variable describing number of customers in the steady state queue.
N_s	Random variable describing number of customers receiving service when the system is in the steady state.
p_n	Steady state probability that there are n customers in the queueing system.
$\pi_q(r),\pi_w(r)$	Symbol for rth percentile queueing time or waiting time; that is, the queueing time or waiting time that r percent of the customers do not exceed.
q,s,w	Random variables describing the time a customer spends in the queue (waiting line), in service, and in the system, respectively; $w = q + s$.
ϱ	Server utilization = traffic intensity/c = $\lambda E[s]/c = (\lambda/\mu)/c$. The probability that any particular server is busy.
τ	Random variable describing interarrival time.
u	Traffic intensity = $E[s]/E[\tau] = \lambda E[s] = \lambda/\mu$. Unit of measure is the erlang.
W	$E[w]$, expected (average or mean) time in the steady state system. $W = W_q + E[s]$.
W_q	$E[q]$, expected (average or mean) time in the queue (waiting line), excluding service time, for steady state system. $W_q = W - E[s]$.

random variables, and we use the symbol τ for an arbitrary interarrival time. The most common arrival pattern in queueing theory terminology is the *random arrival pattern* or *Poisson arrival process*. This means the interarrival time distribution is exponential, that is, $P[\tau \leqslant t] = 1 - e^{-\lambda t}$ for each interarrival time, and the probability of n arrivals in any time interval of length t is $e^{-\lambda t}(\lambda t)^n/n!$, where $n = 0,1,2,\ldots$. Here λ is the average arrival rate, and the number of arrivals per unit time has a Poisson distribution.

Service time distribution. Let s_k be the service time required by the kth arriving customer. In this article, the s_k are assumed to be independent, identically distributed random variables. Therefore, we can refer to an arbitrary service time as s. We also assume the common distribution function $W_s(t) = P[s \leqslant t]$ of service time for all customers. The most common service-time distribution in queueing theory is exponential, which defines the service called *random service*. The symbol μ is reserved for average service rate, and the distribution function for random service is given by $W_s(t) = 1 - e^{-\mu t}$, where $t \geqslant 0$. Other common service time distributions are Erlang-k, hyperexponential, and constant. (See Allen[7] for a discussion of the above probability distributions.)

A statistical parameter that is useful as a measure of the character of probability distributions for interarrival time and for service time is the *squared coefficient of variation* C_X^2, which is defined for a random variable X by

$$C_X^2 = \frac{\text{Var}[X]}{E[X]^2}$$

If X is constant, then $C_X^2 = 0$; if X has an Erlang-k distribution, then $C_X^2 = 1/k$; if X has an exponential distribution, then $C_X^2 = 1$; and if X has a hyperexponential distribution, then $C_X^2 \geqslant 1$. (These results are shown in Allen.[7]) We conclude that, for C_τ^2 nearly equal to zero, the arrival process has a regular pattern; if C_τ^2 is nearly equal to 1, the arrival process is nearly random in character; and if C_τ^2 is greater than 1, arrivals tend to cluster. Similar statements can be made about the service time distribution, where small values of C_s^2 correspond to nearly constant service times and large values correspond to great variability in service times.

Maximum queueing system capacity. In some queueing systems, the queue capacity is assumed to be infinite. That is, every arriving customer is allowed to wait until service can be provided. Other systems, called loss systems, have zero waiting line capacity. That is, if a customer arrives when the service facility is fully utilized, the customer is turned away. Still other queueing systems have a positive (but not infinite) capacity.

Number of servers. The simplest queueing system is the *single-server system*, which can serve only one customer at a time. A *multiserver system* has c identical servers and can serve as many as c customers simultaneously.

Queue discipline. The queue discipline, sometimes called service discipline, is the rule for selecting the next customer to receive service. The most common queue discipline is "first come, first served," abbreviated as FCFS. (Among the whimsical queue disciplines are BIFO for "biggest in first out" and FISH for "first in still here." The reader is probably aware of installations which utilize these queue disciplines.)

A shorthand notation, called the Kendall notation, has been developed to specify queueing systems and has the form *A/B/c/K/m/Z*. Here *A* specifies the interarrival time distribution, *B* the service time distribution, *c* the number of servers, *K* the system capacity, *m* the number in the source, and *Z* the queue discipline. More often a shorter notation, *A/B/c*, is used when there is no limit on the waiting line, the source is infinite, and the queue discipline is FCFS. The symbols used for *A* and *B* are *GI*, general independent interarrival time; *G*, general service time; E_k, Erlang-k interarrival or service time distribution; *M*, exponential interarrival or service time distribution; *D*, deterministic (constant) interarrival or service time distribution; and H_k, hyperexponential (with k stages) interarrival or service time distribution.

Traffic intensity. Traffic intensity u is the ratio of the mean service time $E[s]$ and the mean interarrival time $E[\tau]$. This ratio is an important parameter of a queueing system and is defined by

$$u = \frac{E[s]}{E[\tau]} = \lambda E[s] = \frac{\lambda}{\mu}$$

The traffic intensity u determines the minimum number of servers that are required to keep up with the incoming stream of customers. Thus, for example, if $E[s]$ is 15 seconds and $E[\tau]$ is 10 seconds, $u = 1.5$ and at least two servers are required. The unit of traffic intensity is the erlang, named after A. K. Erlang, a pioneer in queueing theory.

Server utilization. Another important parameter is the traffic intensity per server or u/c, called *server utilization* ϱ when the traffic is evenly divided among the servers. Server utilization is the probability that any given server is busy, and thus, by the Law of Large Numbers, ϱ is the approximate fraction of time that every server is busy. For single-server systems note that $\varrho = u =$ traffic intensity.

Probability that n customers are in the system at time t. This probability, $p_n(t)$, depends not only on t, but also on the initial conditions of the queueing system—that is, the number of customers present when the service facility starts up—and on the other distributions and parameters listed above. For the most useful queueing systems, as t increases, $p_n(t)$ approaches a steady-state value p_n, which is independent of both t and the initial conditions. The system is then said to be in a steady-state condition. This article considers only steady-state solutions to queueing problems because time-dependent or transient solu-

tions are usually too complex for practical use and because the preferred steady-state solution exists in most cases of interest. See Giffin[9] for the transient solutions for some simple queueing models.

Queueing theory provides statistical measures of queueing system performance. Some useful statistical measures (see Table 1) include W_q, W, L_q, and L—the mean queueing time, system time, number in the queue, and number in the system, respectively.

The following formulas, both of which are called Little's law, are quite useful in relating the four primary performance measures:

$$L_q = \lambda W_q$$
$$L = \lambda W$$

Another useful performance measurement is the 90th percentile value of the time in the system, $\pi_w(90)$, which is defined as the amount of time such that 90 percent of all arriving customers spend not more than this amount of time in the system. Expressed symbolically, $\pi_w(90)$ is defined by the equation $P[w \leq \pi_w(90)] = 0.9$. The 90th percentile value of time in queue, $\pi_q(90)$, is similarly defined.

Applications of a simple open queueing model

The *M/M/c* queueing system is a simple, open, queueing model that can be used to model, at least approximately, many computer systems. It is considered open because customers enter the system from outside, receive service, and leave the system. Closed systems, in which customers never leave the system, will be considered later. The equations for the *M/M/c* queueing system are shown in Table 2. (More extensive equations for this system are given in Reference 7.)

Example 1. A branch office of Endearing Engineering, an engineering consulting firm, has one on-line terminal connected to a central computer system for eight hours each day. Engineers, who work throughout the city, drive to the branch office to make routine calculations. Their arrival pattern is random (Poisson) with an average of 10 persons per day using the terminal. The distribution of time spent by an engineer at the terminal is approximately exponential with an average value of half an hour. Thus the terminal is 5/8 utilized ($10 \times 1/2 = 5$ hours out of 8 hours available). The branch manager receives complaints from the staff about the length of time many of them have to wait to use the terminal. It does not seem reasonable to the manager to procure another terminal when the present one is used only five-eighths of the time, on the average. How can queueing theory help this manager?

Solution. The *M/M/1* queueing system is a reasonable model of this system with $\varrho = 5/8$ as computed above. (There actually are only a finite number of engineers but the infinite population assumption seems reasonable here.) Then, using the equations from Table 2 for the *M/M/c* model with $c = 1$, we can compute the standard performance measures.

$$W_q = \frac{\varrho E[s]}{1-\varrho} = 50 \text{ minutes}$$
Average time an engineer spends in the queue

$$W = \frac{E[s]}{1-\varrho} = 80 \text{ minutes}$$
Average time an engineer spends at the branch office waiting for and using the terminal

$$\pi_q(90) = W \ln(10\varrho) = 146.61 \text{ minutes}$$
90th percentile queueing time

Also, since λ is 10 engineers per eight-hour day, which is

$$\frac{10}{(8 \times 60)} = \frac{1}{48} \text{ engineers per minute}$$

we can use Little's law to calculate

$$L_q = \lambda W_q = 1.0417$$
Average number of engineers in the queue

and

$$L = \lambda W = 1.667 \text{ engineers}$$
Average number of engineers in the branch office to use the terminals

Table 2.
Steady-state formulas for *M/M/c* queueing system.

$u = \lambda/\mu = \lambda E[s]$	Traffic intensity
$\varrho = u/c$	Server utilization
$C(c,u)$	The probability all c servers are busy so that an arriving customer must wait; can be calculated by Erlang's C formula (below)
$C(c,u) = \dfrac{\frac{u^c}{c!}}{\frac{u^c}{c!} + (1-\varrho)\sum_{n=0}^{c-1}\frac{u^n}{n!}}$	Erlang's C formula
$W_q = \dfrac{C(c,u)E[s]}{c(1-\varrho)}$	Mean queueing time
$W = W_q + E[s]$	Mean time in the system
$\pi_q(90) = \dfrac{E[s]}{c(1-\varrho)} \ln(10\,C(c,u))$	90th percentile time in the queue
When $c = 1$, the *M/M/1* formulas simplify to	
$C(c,u) = \varrho = \lambda E[s]$	
$W_q = \dfrac{\varrho E[s]}{1-\varrho}$	
$W = \dfrac{E[s]}{1-\varrho}$	
and	
$\pi_q(90) = W \ln(10\varrho)$.	

These statistics show that slightly more than one engineer-day is being lost by engineers queueing up to use the terminal. In the next example we will see how the *M/M/c* queueing system model can be used to make an informed decision on how to solve the problem.

Example 2. In Example 1 we discovered a puzzling situation at Endearing Engineering. Although their remote terminal was only 62.5 percent utilized, the average time an engineer had to queue for the terminal was 50 minutes with 10 percent of them having to wait for over 146.61 minutes. A committee of engineers met with the branch manager and showed how queueing theory could explain what was happening. They decided the problem could not be solved by scheduling terminal time; one or more additional terminals should be provided. They specified that the mean queueing time should not exceed 10 minutes with the 90th percentile value of queueing time not to exceed 15 minutes. The branch manager then had second thoughts; he reasoned that, if the average queueing time is 50 minutes with one terminal, then it must be 25 minutes with two terminals, and thus five terminals would be required to make $W_q \leq 10$ minutes. How many terminals *are* required?

Solution. The solution, of course, depends upon whether all the terminals are installed at the branch office, giving one *M/M/c* queueing system, or are distributed to several customer locations, thus providing multiple single-server (*M/M/*1) queueing systems. Let us consider the former case first, by trying c = 2, that is, providing two terminals in the branch office. (We are motivated, in part, by the fact that the equations are much easier to solve for $c = 2$ than for $c = 5$.)

$$\varrho = 0.625/2 = 0.3125 \qquad \text{Server utilization}$$

$$C(2,u) = C(2,0.625) = 0.1488 \qquad \text{Probability both servers are busy}$$

$$W_q = \frac{C(c,u)E[s]}{c(1-\varrho)} = 3.247 \text{ minutes} \qquad \text{Mean time in the queue}$$

$$\pi_q(90) = \frac{E[s]}{c(1-\varrho)} \ln(10\,C(c,u)) = 8.67 \text{ minutes} \qquad \text{90th percentile queueing time}$$

Thus one additional terminal in the branch office will satisfy all the requirements.

If the additional terminal is placed at a customer location and the traffic to the two terminals is evenly split, there would be two *M/M/*1 queueing systems, each with $\varrho = 0.3125$. Then, by the formulas of Table 2, for $c = 1$, we calculate

$$W_q = \frac{\varrho E[s]}{1-\varrho} = 13.64 \text{ minutes} \qquad \text{Mean queueing time}$$

$$\pi_q(90) = \frac{E[s]}{1-\varrho} \ln(10\varrho) = 49.72 \text{ minutes.} \qquad \text{90th percentile queueing time}$$

Thus two distributed terminals will not meet either of the criteria.

Table 3 shows the results for various numbers of terminals. Some, rising above principle, might decide to settle for four distributed terminals; however, the stated criteria mean that, if the terminals are dispersed, five are required. Thus the branch manager is right. (It's not nice to fool your manager!) We have not, of course, considered the travel time for engineers to reach a terminal. Using queueing theory models and cost information concerning travel time to the terminals, it is possible to determine the most cost effective solution to the problem. (For an example, see Reference 7, Chapter 5, Exercise 13.)

Finite population queueing models of interactive computer systems

In the two previous examples we used an open queueing system model with an infinite number of customers. Actually, there are few, if any, real-life queueing systems that are truly open and infinite; however, many systems can be reasonably approximated by such models as explained by Buzen and Goldberg.[8] We will now consider a more realistic queueing model which is closed (no customer enters or leaves the system) and has a finite customer population.

Figure 5 portrays what is commonly known as a finite population queueing model of an interactive computer system (see Muntz[10] and Allen[7]). The central processor system consists of (possibly) a queue for the central processor system, a CPU, (perhaps) some I/O devices, and an associated system of queues for these service features. The customers (users) interact with the central processor system through N terminals. Each customer (user) is assumed to be in exactly one of three states at any instant of time: (1) "thinking" at the terminal (this time is called "think time, t"), (2) queueing for some type of service, or (3) receiving service. Think time, t, includes all the elapsed time between the completion of service at the central processor system for an interaction until a request is submitted for another interaction. Kobayashi,[11] who provides an excellent discussion of this system, uses the name "user time" for what we call "think time." A user at a terminal cannot submit a new request for CPU service until the previous request has been satisfied. In Figure 5 the customer (user) can be represented as a token which circulates around the system and which at any instant is either at a terminal, at the CPU, or at a queue in the central processor system. The particular queueing network we study is determined by the model we select for the central processor system. The equations valid for all models of the type shown in Figure 5 are given in Table 4; they describe the model under very general

conditions (see Kobayashi[11]). The value of p_0 usually depends upon the distributional form of both the think time and the CPU service time. Thus the values of W and λ_T are determined by the model we use for the central processor system.

For studying timesharing systems one useful model of the central processor system is that where the CPU service time is general and the processor-sharing queue discipline is employed. This model is discussed in considerable detail by Kleinrock,[12] who played a significant role in developing it. The processor-sharing queue discipline assumes the processing power of the CPU is divided equally among all requests for service. Thus, if there are n interactions pending for CPU service, each of them is instantly served at the rate of μ/n customers per unit time, where μ is the CPU processing rate. This queue discipline is derived as the limiting case of a round-robin timesharing system as the quantum of service provided each user approaches zero. In practice it gives good results for quantized service times with finite quanta which are short compared with the usual service time requirement. If the central processor system of Figure 5 is replaced by a single CPU utilizing the processor-sharing queue discipline, then the equations of Table 4 apply with p_0 calculated by the formula

$$\text{(A)} \qquad p_0 = \left[\sum_{n=0}^{N} \frac{N!}{(N-n)!}\left(\frac{E[s]}{E[t]}\right)^n\right]^{-1}$$

We also can calculate the following:

$$\text{(B)} \qquad \varrho = 1 - p_0 \qquad \text{CPU utilization}$$

$$\text{(C)} \qquad \lambda_T = \frac{\varrho}{E[s]} \qquad \text{Throughput in interactions per unit time}$$

Example 3. We consider one of the examples used by Martin Reiser[13] in his interesting article explaining how a sophisticated APL program called QNET4, which was developed at the IBM Research Division, can be used to model computer systems. A finite processor-sharing interactive computer system has 20 active terminals, a mean think time of 3 seconds, a CPU average service rate of 500,000 instructions per second (0.5 MIPS), and an average interaction requirement of 100,000 instructions. It is desired to find the mean response time, W; the average throughput, λ_T; the CPU utilization; and the average number of interactions pending in the CPU system. We would also like to know how this would change if we added 10 more terminals so there would be 30 in all.

Solution. Since each interaction requires the execution of an average of 100,000 CPU instructions, we have

$$E[s] = 100{,}000/500{,}000 = 0.2 \text{ seconds}$$

By Equation (A) above, p_0, the probability the CPU is idle, is given by

$$p_0 = \left[\sum_{n=0}^{N} \frac{N!}{(N-n)!}\left(\frac{E[s]}{E[t]}\right)^n\right]^{-1} = 0.045593216$$

Then Equation (1) of Table 4 yields the mean response time W as

$$W = \frac{NE[s]}{1-p_0} - E[t] = 1.191 \text{ seconds}$$

The CPU utilization, ϱ, is given (by Equation (B) above) as

$$\varrho = 1 - p_0 = 0.9544$$

Table 3.
Summary of Example 2 calculations.
(All times in minutes)

TERMINALS	SYSTEM	ϱ	W_q	$\pi_q(90)$
1	*M/M/1*	0.62500	50.00	146.61
1	*M/M/2*	0.31250	3.25	8.67
2	*M/M/1's*	0.31250	13.64	49.72
4	*M/M/1's*	0.15625	5.56	15.87
5	*M/M/1's*	0.12500	4.29	7.65

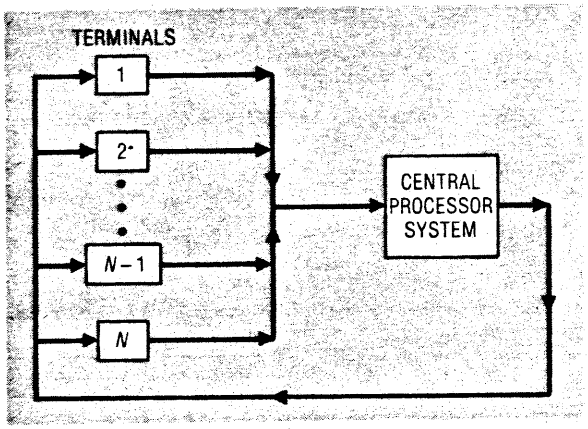


Figure 5. Finite population queueing model of interactive computer system. (© Academic Press, *Probability, Statistics, and Queueing Theory with Computer Science Applications*, 1978)

Table 4.
Finite population queueing model of interactive computer system depicted in Figure 5.

The equations which describe this model are

(1) $W = \dfrac{NE[s]}{1-p_0} - E[t]$ — Mean response time

(2) $\lambda_T = \dfrac{N}{W+E[t]}$ — Throughput in interactions per unit time

In the equations

$E[s]$ = mean CPU service time per interaction,

p_0 = probability the system is idle, and

$E[t]$ = mean "think time" (the mean time between successive requests at a terminal).

This yields the average throughput, λ_T, given (by Equation (C) above) as

$$\lambda_T = \frac{\varrho}{E[s]} = 4.722 \text{ interactions per second}$$

By Little's law, the average number of interactions pending in the CPU is

$$L = \lambda_T W = 5.68 \text{ interactions}$$

If the number of terminals is raised to 30, then, by the equations we used before, we get the following results:

$$\begin{aligned} p_0 &= 0.00022118 \\ \varrho &= 0.99977882 \\ W &= 3.00 \text{ seconds} \\ \lambda_T &= 5 \text{ interactions per second} \\ L &= 15 \text{ interactions} \end{aligned}$$

Thus a 50 percent increase in number of terminals increased the throughput only 4.78 percent while increasing the response time by 151.9 percent!

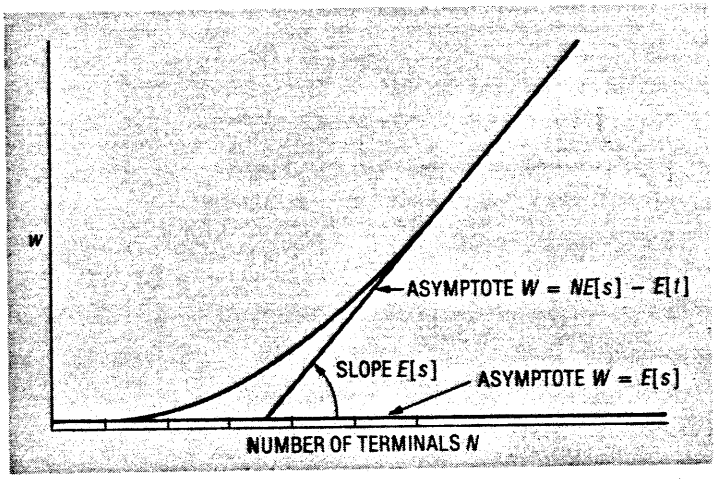


Figure 6. Mean response time, *W*, versus *N*, the number of terminals, for the finite population queueing model of an interactive computer system. (© Academic Press, *Probability, Statistics, and Queueing Theory with Computer Science Applications*, 1978)

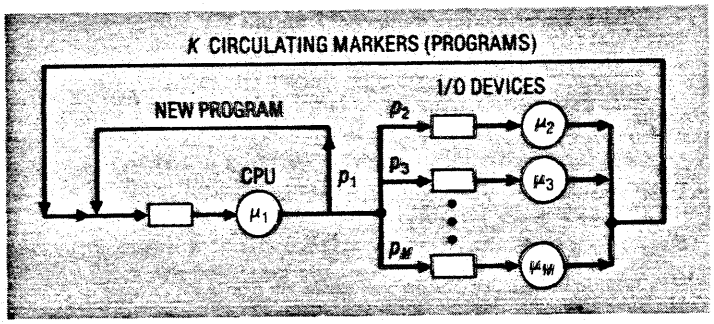


Figure 7. Central server model of multiprogramming. (© Academic Press, *Probability, Statistics, and Queueing Theory with Computer Science Applications*, 1978)

This example illustrates the concept of system saturation. Consider Figure 6, the graph of the mean response time

$$W = \frac{NE[s]}{1-p_0} - E[t]$$

versus N. For $N = 1$ there is no queueing so $W = E[s]$. For small values of N the customers interfere with each other very little; that is, when one person wants a CPU interaction, the others are usually in think mode so little queueing occurs. Thus the curve is asymptotic at $N = 1$ to the line $W = E[s]$. As $N \to \infty$, $p_0 \to 0$ since the likelihood of the CPU being idle must go to 0. Hence the curve is asymptotic to the line $NE[s] - E[t]$ as $N \to \infty$. The two asymptotes intersect where

$$N = N^* = \frac{E[s]+E[t]}{E[s]}$$

Kleinrock[12] calls N^* the *system saturation point*. He points out that, if each interaction required exactly $E[s]$ units of CPU service time and exactly $E[t]$ units of think time, then N^* is the maximum number of terminals that could be scheduled in such a way as to cause no mutual interference. For $N << N^*$, there is almost no mutual interference and W is approximately $E[s]$. For $N >> N^*$, users "totally interfere" with each other; that is, the addition of a terminal raises everyone's average response time by $E[s]$. In Example 3, $N^* = 3.2/0.2 = 16$ terminals, and the increase in W due to the change from 20 terminals to 30 terminals was close to $10 \times 0.2 = 2$ seconds. (Actually it was 1.809 seconds.)

The central server model of multiprogramming

A number of models have been formulated to represent the central processor system in Figure 5. One that has had a great deal of success in practice is the central server model. It also has been used to model batch multiprogramming systems. It is more complex (and thereby more realistic) than any of the previous models we have considered.

This model, shown in Figure 7, is a closed model since it contains a fixed number of programs K which can be thought of as markers which cycle around the system interminably. However, each time a marker (program) makes the cycle from the CPU directly back to the CPU, we assume a program execution has been completed and a new program enters the system. It is sometimes convenient to consider the model of Figure 7 to represent the central processor system of a more complex system such as that in Figure 5, but we must then assume the users at the terminals are sufficiently active to guarantee a pending interaction. Otherwise the two models are incompatible.

There are $M-1$ I/O devices, each with its own queue and each having exponentially distributed ser-

vice times with average service rate μ_i (i = 2,3,...,M). The CPU also is assumed to provide exponential service (with average rate μ_1). Upon completion of a CPU execution, the job returns to the CPU (completes execution) with probability p_1 or requires service at I/O device i with probability p_i, i = 2,3,...,M. Upon completion of I/O service the job returns to the CPU queue for another cycle. If we let $\mathbf{k} = (k_1,k_2,\ldots,k_M)$ represent the state of the system, where k_i is the number of jobs (markers) at the ith queue (queueing or in service), then Buzen [14,15] shows the probability $p(k_1,k_2,\ldots,k_M)$ that the system is in state $\mathbf{k}$ is given by

$$p(k_1,k_2,\ldots,k_M) = \frac{1}{G(K)} \prod_{i=2}^{M} \left(\frac{\mu_1 p_i}{\mu_i}\right)^{k_i}$$

for any $(k_1,k_2,\ldots,k_M)$ such that

$$\sum_{i=1}^{M} k_i = K$$

and each $k_i \geq 0$, where $G(K)$ is defined so as to make the probabilities sum to 1. (In Figure 7 the rectangles represent queues and the circles represent service facilities.) A technique developed by Buzen for calculating $G(0) = 1$, $G(1)$, $G(2)$,...,$G(K)$ is given in Table 5. The equations to describe the central server model are given in Table 6.

Example 4. Mr. G. Nee Uss, the senior computer system performance analyst at Saggin Waggin, a subsidiary of Admiral Motors, decides to model their main on-line computer system using the finite population queueing system model of Figure 5 with the central processor system represented by the central server model. During the busiest period of the day they have an average of 100 active terminals with a mean think time of 13 seconds. Their observed mean response time is 6.41 seconds. By Formula (2) of Table 4, we see that

$$\lambda_T = \frac{N}{W + E(t)} = 5.15$$

so that, by Little's law,

$$L = \lambda_T W = 33$$

interactions are pending, on the average. Thus the activity at the terminals is certainly great enough to ensure compatibility of the finite population and central server models. The parameters of the central server part of their model are

$M = 3$	One CPU and two I/O devices
$K = 4$	Multiprogramming level is four
$\mu_1 = 100$ $\mu_2 = 25$ $\mu_3 = 40$	Service rates
$p_1 = 0.1$ $p_2 = 0.2$ $p_3 = 0.7$	Branching probabilities

Table 5.
Buzen's algorithm.

Given the parameters of the central server model of Figure 7 (that is, μ_i, p_i for $i = 1,2,\ldots,M$), this algorithm will generate $G(K)$ as well as $G(K-1)$, $G(K-2),\ldots,G(1),G(0) = 1$.

Step 1, assign values to x_i:
Set $x_1 = 1$ and then set $x_i = \mu_1 p_i/\mu_i$ for $i = 2,3,\ldots,M$.

Step 2, set initial values:
Set $g(k,1) = 1$ for $k = 0,1\ldots,K$ and set $g(0,m) = 1$ for $m = 1,2,\ldots,M$

Step 3, initialize k:
Set k to 1.

Step 4, calculate kth row:
Set $g(k,m) = g(k,m-1) + x_m g(k-1,m)$, $m = 2,3,\ldots,M$.

Step 5, increase k:
Set k to $k + 1$.

Step 6, algorithm complete?
If $k \leq K$, return to Step 4. Otherwise, terminate the algorithm.
Then $g(n,M) = G(n)$ for $n = 0,1,\ldots,K$.

Observed values of the respective utilizations are

$\varrho_1 = 0.521$	CPU utilization
$\varrho_2 = 0.409$	First I/O device utilization
$\varrho_3 = 0.911$	Second I/O device utilization
$\lambda_T = 5.18$ interactions/sec	Mean throughput
$W = 6.41$ sec	Mean response time

If the proposed model seems to fit reasonably well, Mr. Uss wants to use it to investigate the effects of two possible hardware upgrades. The first upgrade considered is the procurement of enough additional main memory to allow the multiprogramming level, K, to be increased from 4 to 15. The second possible

Table 6.
Steady-state equations of central server model of multiprogramming (see main text for model assumptions).

Calculate $G(0),G(1),\ldots,G(K)$ by Buzen's algorithm.

Then the server utilizations are given by

$$(1) \qquad \varrho_i = \begin{cases} G(K-1)/G(K), & i = 1 \\ \dfrac{\mu_1 \varrho_1 p_i}{\mu_i}, & i = 2,3,\ldots,M \end{cases}$$

The average throughput, λ_T, is given by

$$(2) \qquad \lambda_T = \mu_1 \varrho_1 p_1.$$

If the central server model is the central processor model for the interactive computing system of Figure 5, then the average response time, W, is calculated by

$$(3) \qquad W = \frac{N}{\lambda_T} - E[t] = \frac{N}{\mu_1 \varrho_1 p_1} - E[t].$$

upgrade is to make hardware and software changes which will effectively speed up both I/O devices by 25 percent while keeping the multiprogramming level at four.

Solution. Applying Buzen's Algorithm we calculate

$$x_1 = 1, x_2 = \mu_1 p_2/\mu_2 = 0.8$$

and

$$x_3 = \mu_1 p_3/\mu_3 = 1.75$$

Continuing with Buzen's algorithm yields the following table for the original model:

	x_1	x_2	x_3	
	1	0.8	1.75	
0	1	1	1	$= G(0)$
1	1	1.8	3.55	$= G(1)$
2	1	2.44	8.6525	$= G(2)$
3	1	2.952	18.093875	$= G(3)$
4	1	3.3616	35.02588125	$= G(4) = G(K)$

Then, by the equations of Table 6,

$\varrho_1 = G(3)/G(4) = 0.51658586$ CPU utilization

$\varrho_2 = \mu_1 \varrho_1 p_2/\mu_2$
$= 0.413268688$ Utilization of first I/O device

$\varrho_3 = \mu_1 \varrho_1 p_3/\mu_3$
$= 0.904025256$ Utilization of second I/O device

$\lambda_T = \mu_1 \varrho_1 p_1$
$= 5.1658586$ interactions per second Mean throughput

$W = \frac{N}{\lambda_T} - E[t]$
$= 6.357866$ seconds Mean response time

These values are certainly close enough to the measured values to validate the model. Mr. Uss should proceed.

Similar calculations show that if the multiprogramming level is increased to 15 from 4, then

$\varrho_1 = 0.571282414$
$\varrho_2 = 0.457025931$
$\varrho_3 = 0.999744225$
$\lambda_T = 5.71282$ interactions per second

and

$W = 4.505$ seconds

Thus, there has been a 10.6 percent improvement in mean throughput, λ_T, and a 29.2 percent decrease in mean response time, W.

If the multiprogramming level is kept at four but the I/O devices are 25 percent faster, Mr. Uss's calculations show similarly that

$\varrho_1 = 0.616300129$
$\varrho_2 = 0.394432082$
$\varrho_3 = 0.86282018$
$\lambda_T = 6.163$ interactions per second

and

$W = 3.226$ seconds

This upgrade seems to be much more favorable than increasing the multiprogramming level to 15. It improves mean throughput by 19.3 percent and decreases the mean response time by 49.3 percent. It also does not overload the I/O devices as heavily as the first upgrade option.

Similar calculations can also show Mr. Uss the improvement to expect if both of the upgrades are done simultaneously, that is, if we get faster I/O devices and enough main memory to raise the multiprogramming level to 15. This would give a great deal of improvement. Mr. Uss calculates that

$\lambda_T = 7.123$ interactions per second

and

$W = 1.039$ seconds

This is a 15.6 percent improvement in mean throughput and a 67.8 percent decrease in the mean response time over the system with faster I/O devices but with the multiprogramming level kept at four. Compared to the original system it shows a 24.7 percent increase in λ_T and a 76.9 percent decrease in W.

The reader should note that what we have glibly represented as an I/O device may very well be represented in physical hardware by a block multiplexer channel with several attached disk drives in the case of the first I/O device, and by such a channel with several drums in the case of the second I/O device.

If the first I/O device on the original system is replaced by one with the same speed as the second I/O device, and the load on the two devices is balanced so that the branching probability to each is 0.45, then we would have

$\lambda_T = 6.14$ interactions per second

and

$W = 3.285$ seconds

The central server model is one of the most successful analytic queueing models in use for modeling multiprogramming computer systems. Price[16] describes how it was successfully used to model an IBM System/360-91 at the Stanford Linear Accelerator Center. Buzen[17,18] describes a number of successful modeling efforts which utilized the central server model.

Some useful approximations

Many computer systems can be modeled as a "network of queues," that is, a network of simple queueing systems in which the input(s) to one queueing system may be output(s) from one or more other queueing system(s). Unfortunately little can be done, analytically, with general queueing networks, except in the simple case covered by Jackson's theorem (see Allen[7]) in which all elements of the network are *M/M/c* queueing systems and all arrival patterns are random (Poisson). However, there are some simple approximations which can be used to model fairly complex queueing networks.

One useful approximation noted by the author and his colleague, John Cunneen, can be stated (modestly) as the *Allen-Cunneen approximation formula:* For any *GI/G/c* queueing system it is approximately true that

$$\text{(D)} \qquad W_q = \frac{C(c,u)E[s]}{c(1-\varrho)} \left\{ \frac{C_\tau^2 + C_s^2}{2} \right\}$$

In this formula, $C(c,u)$ is Erlang's C formula (shown in Table 2); C_τ^2, C_s^2 is the squared coefficient of variation for the interarrival time and service time, respectively; and ϱ is the server utilization. A similar, but slightly more complex formula for the single server case has been given by Kuehn.[19] The approximation formula above is exact for *M/M/c* and *M/G/1* queueing systems and gives a reasonably good approximation for many others. Examples of its use are given in Allen.[7] It is easy to compute, and the further approximation

$$\text{(E)} \qquad \frac{C(c,u)}{c(1-\varrho)} \approx \frac{\varrho^c}{1-\varrho^c}$$

can be used to make the computation even easier. Equation (E) is exact for $c = 1,2$ but is a little low for other values of c.

Sevcik et al.[20] give the approximations shown in Figure 8 for computing the λ and C_τ^2 values for departures, splits, and merges in a queueing network. Then the Allen-Cunneen approximation can be used to calculate values of W_q and thereby W, L_q, and L. We illustrate in the following example.

Example 5. Consider the computer system model of Figure 9, taken from Sevcik et al.[20] We solve this model iteratively by assuming starting values for C_τ^2 at each queueing facility. Then we solve for λ_1 and $\lambda_2 = \lambda_3 = \lambda_1/2$ by using Little's law

$$4 = \sum_{i=1}^{3} \lambda_i W_i$$

where each

$$W_i = W_{q_i} + E[s_i]$$

and the Allen-Cunneen approximation is used to calculate each W_{q_i}. The formulas in Figure 8 are then

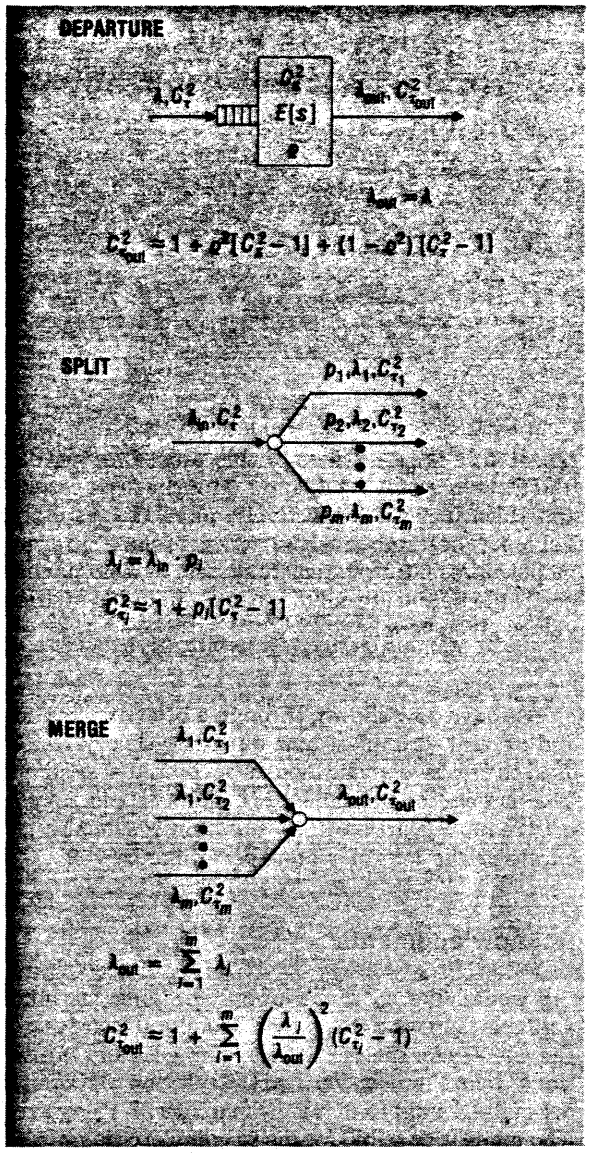


Figure 8. Approximations for queueing networks.

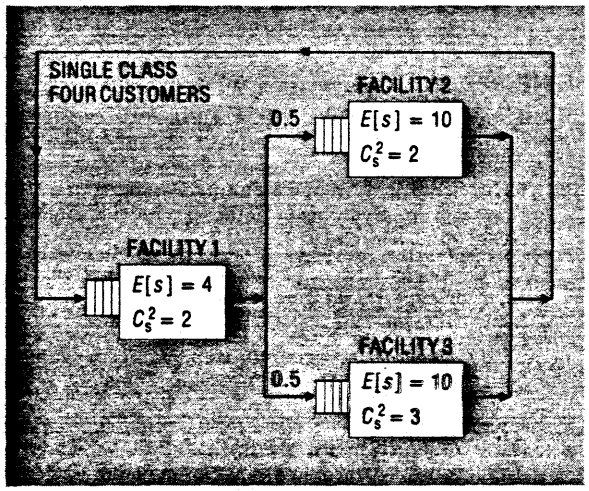


Figure 9. Queueing network of Example 5.

Table 7.
Results of Example 5.

	FACILITY.		
	1	2	3
λ	0.1037	0.0518	0.0518
ϱ	0.4147	0.5184	0.5184
W_q	4.6300	17.2200	22.6000
W	8.6300	27.2200	32.6000
L_q	0.4804	0.8927	1.1716
L	0.8951	1.4111	1.6900
C_τ^2	1.2700	1.2000	1.2000

used to recompute the C_τ^2 values and the process is repeated. Four iterations of this procedure with initial C_τ^2 values of 2.5, 1.6, and 1.6 for the first, second, and third facility, respectively, gave the results in Table 7. (The numbers given in Sevcik et al.[20] are for a model representing the entire system by a single composite queue and are not directly comparable with the values given here.)

We have now shown, through a brief explanation and several examples, how queueing theory can be used to create analytic models of computer systems. The models thus obtained are efficient and easy to use, requiring little effort to obtain reasonably accurate predictions of system performance. ■

Acknowledgments

I would like to thank John Spragins for much valuable help with this paper. Thanks also are due to the referees for their excellent comments and suggestions. John Cunneen verified the solution to Example 5. Finally, I want to thank my manager, J. Perry Free, who made it possible for me to complete this work.

References

1. P. Dickson, *The Official Rules,* Delacorte Press, New York, 1978.
2. R. M. Schardt, "An MVS Tuning Perspective," IBM Washington Systems Center Technical Bulletin, GG22-9023, March 1979.
3. H. C. Lucas, Jr., "Performance Evaluation and Monitoring," *ACM Computing Surveys,* Vol. 3, No. 3, Sept. 1971, pp. 79-91.
4. *TPNS General Information Manual,* Form Number GH20-1907, IBM Data Processing Division, White Plains, New York.
5. *ACM Computing Surveys,* Vol. 10, No. 3, Sept. 1978.
6. J. Spragins, "Approximate Techniques for Modeling the Performance of Complex Systems," *Computer Languages,* Vol. 4, No. 2, 1979, pp. 99-129.
7. A. O. Allen, *Probability, Statistics, and Queueing Theory With Computer Science Applications,* Academic Press, New York, 1978.
8. J. P. Buzen and P. S. Goldberg, "Guidelines for the Use of Infinite Source Queueing Models in the Analysis of Computer System Performance, *AFIPS Conf. Proc.,* 1974 NCC, pp. 371-374.
9. W. C. Giffin, *Queueing,* Grid, Inc., Columbus, Ohio, 1978.
10. R. Muntz, "Analytic Modeling of Interactive Systems," *Proc. IEEE,* Vol. 63, No. 6, June 1975, pp. 946-953.
11. H. Kobayashi, *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology,* Addison-Wesley, Reading, Mass., 1978.
12. L. Kleinrock, *Queueing Systems Volume 2: Computer Applications,* John Wiley, New York, 1976.
13. M. Reiser, "Interactive Modeling of Computer Systems," *IBM Systems J.,* Vol. 15, No. 4, 1976, pp. 309-327.
14. J. P. Buzen, "Computational Algorithms for Closed Queueing Networks with Exponential Servers," *Comm. ACM,* Vol. 16, No. 9, Sept. 1973, pp. 527-531.
15. J. P. Buzen, *Queueing Network Models of Multiprogramming,* PhD Thesis, Division of Engineering and Applied Physics (NTIS AD 731 575, August 1971), Harvard University, Cambridge, Mass., May 1971.
16. T. G. Price, Jr., "A Comparison of Queueing Network Models and Measurements of a Multiprogrammed Computer System," *ACM Performance Evaluation Review,* Vol. 5, No. 4, Fall 1976, pp. 39-62.
17. J. P. Buzen, "Modelling Computer System Performance," *CMG VII Conf. Proc.,* Atlanta, Georgia, Nov. 1976.
18. J. P. Buzen, "A Queueing Network Model of MVS," *ACM Computing Surveys,* Vol. 10, No. 3, Sept. 1978, pp. 319-331.
19. P. J. Kuehn, "Approximate Analysis of General Queueing Networks by Decomposition," *IEEE Trans. Comm.,* Vol. COM-27, No. 1, Jan. 1979, pp. 113-126.
20. K. C. Sevcik, A. I. Levy, S. K. Tripathi, and J. L. Zahorjan, "Improving Approximations of Aggregated Queueing Network Subsystems," in *Computer Performance,* K. M. Chandy and M. Reiser, eds., North Holland, New York, 1977, pp. 1-22.

Arnold O. Allen is a senior instructor at the Los Angeles IBM Systems Science Institute where he is responsible for a customer class in computer system performance evaluation and capacity planning; he also teaches other data processing management classes. Previous assignments at IBM include programming, system engineering, marketing, and recruiting.

He is a member of Phi Beta Kappa, Sigma Xi, ACM, IEEE, SIAM, ASA, and ORSA. He holds MA and PhD degrees in mathematics from UCLA.